

TP6 et 7 – Système à base de règles en Prolog

Chloé Conrad, Nathalie Guin, Marie Lefevre & Maëva Somny

SYSTEMES A BASE DE REGLES

On considère des problèmes que l'on peut résoudre à l'aide d'un système à base de règles. Nous allons, dans un premier temps, définir un moteur d'inférences à chaînage avant que nous appliquerons sur un problème jouet. Vous définirez ensuite la base de règles relative à un problème de météo. Attention, votre moteur doit être générique et fonctionner sur le problème jouet, le problème de météo mais également sur tout autre jeu de test (une autre base de règles).

PARTIE 1 : MOTEUR EN CHAÎNAGE AVANT

Notre problème jouet est décrit comme suit. Nous disposons de 6 règles :

- R1 : $A \text{ et } B \rightarrow C$
- R2 : $C \text{ et } \text{non}(D) \rightarrow F$
- R3 : $F \text{ et } B \rightarrow E$
- R4 : $F \text{ et } A \rightarrow \text{non}(G)$
- R5 : $\text{non}(G) \text{ et } F \rightarrow B$
- R6 : $A \text{ et } H \rightarrow L$

Nous connaissons trois faits : A , C et $\text{non}(D)$, et nous voulons démontrer le fait E .

Nous allons réaliser un moteur d'inférences en chaînage avant qui fonctionne comme sur l'exemple suivant :

```
?- raz.                                /* on vide la base de faits */
true.

?- faits([a,c,non(d)]). /* on charge les nouveaux faits */
true.

?- saturer.                            /* on lance le moteur en chainage avant */
r2 : non(d) a c f
r4 : non(d) non(g) a c f
r5 : non(d) non(g) a c f b
r1 : non(d) non(g) a c f b
r3 : non(d) non(g) a c f b e
true.
```

Pour réaliser ce moteur d'inférences en chaînage avant, il faut :

Question 1 : Définir la base de règles sous forme de listes composées de listes de prémisses et de listes de conclusions, tel que :

```
regle(Ri, ListeDePrémisses, ListeDeConclusions).
```

Question 2 : Définir un prédicat permettant à l'utilisateur d'initialiser la base de faits. On utilisera le prédicat `assert` pour ajouter `vrai(Fait)` pour les faits positifs et `faux(Fait)` pour les faits négatifs.

Question 3 : Définir un prédicat permettant à l'utilisateur de vider la base de faits. On utilisera le prédicat `retractall` pour supprimer les faits.

Question 4 : Définir un prédicat `saturer` qui sature la base de règles et produit une trace de son fonctionnement. L'algorithme utilisé pour ce moteur sera le suivant :

```

Changement <-- Vrai
Tant que Changement est Vrai
    Changement <-- Faux
    Boucle sur les règles : soit R une règle de BaseRègles
        Si R n'est pas marquée
        et si les prémisses de R appartiennent à BaseFaits
        Alors ajouter les conclusions de R à BaseFaits
            changement <-- Vrai
            marquer R
        FinSi
    FinBoucle
FinTantQue

```

Rappel 1 : La seule boucle qui existe en Prolog est la boucle mue par l'échec.

Rappel 2 : Lorsque vous voulez utiliser un prédicat relatant un fait (par exemple `vrai/1`) dans un autre prédicat, mais que votre base de faits n'est pas encore remplie, vous devez le déclarer dynamiquement :
`:- dynamic vrai/1, faux/1.`

PARTIE 2 : NOUVELLE BASE DE CONNAISSANCES

Question 5 : Tester votre moteur d'inférences sur la base de règles ci-dessous :

R1 : s'il fait beau aujourd'hui alors il fera beau demain
 R2 : s'il pleut aujourd'hui alors il pleuvra demain
 R3 : si le vent est d'ouest alors le temps est à la pluie
 R4 : s'il y a des nuages alors le temps est à la pluie
 R5 : si les araignées tissent des toiles lâches alors les animaux pressentent la pluie
 R6 : si les hirondelles volent bas alors les animaux pressentent la pluie
 R7 : si les animaux pressentent la pluie alors il pleuvra demain
 R8 : s'il pleut en Bretagne et le vent est d'ouest alors le temps est à la pluie
 R9 : si le temps est à la pluie alors il pleuvra demain
 R10 : s'il fait beau demain alors il pleuvra demain

Question 6 : Qu'obtenez-vous en chaînage avant avec la base de faits suivante :

- il fait beau aujourd'hui
- il y a des nuages
- les araignées font des toiles lâches
- le vent est d'ouest

PARTIE 3 : MOTEUR EN CHAINAGE ARRIERE

Question 7 : Écrire un moteur d'inférences d'ordre 0 fonctionnant en chaînage arrière. On représentera la base de règles et la base de faits comme dans la première partie.

Pour rappel, le chaînage arrière consiste à assigner un but au système et à unifier la partie conclusion des règles avec ce but. Les nouveaux buts à démontrer sont alors les prémisses de la règle sélectionnée. Si les prémisses ne sont pas dans la base de faits, ni démontrables *via* une autre règle, vous ferez échouer la démonstration.

Sur notre problème jouet, son exécution pourra donner par exemple :

```
?- raz.                                /* on vide la base de faits */
true.

?- faits([a,c,non(d)]).                /* on charge les nouveaux faits */
true.

?- satisfait(e) .                      /* on cherche à démontrer E */
c dans la base de faits
non(d) dans la base de faits
f satisfait grace a r2
c dans la base de faits
non(d) dans la base de faits
f satisfait grace a r2
a dans la base de faits
non(g) satisfait grace a r4
c dans la base de faits
non(d) dans la base de faits
f satisfait grace a r2
b satisfait grace a r5
e satisfait grace a r3
true

?- satisfait(l).                      /* on cherche à démontrer L */
a dans la base de faits
false.
```

Question 8 : Qu'obtenez-vous avec la base de règles de la météo, la base de faits de la question 6 et chacun de deux buts suivants :

- il fera beau demain
- il pleuvra demain

PARTIE 4 : MOTEUR A CHAINAGE MIXTE

Nous vous proposons d'effectuer une amélioration du fonctionnement du moteur d'inférences en mettant en œuvre un chaînage mixte selon la description ci-dessous.

On distingue deux catégories de faits :

- les faits « terminaux », qui ne figurent jamais en partie gauche d'une règle,
- les faits « observables », qui ne figurent jamais en partie droite d'une règle.

Question 9 : Définir le prédicat `terminal(F)` (respectivement `observable(F)`) qui est vrai si le fait `F` est terminal (resp. observable).

Il est possible que la base de faits fournie par l'utilisateur ne permette pas, en inférant en chaînage avant, de conclure sur un fait terminal. Dans ce cas, on ne peut pas répondre à l'utilisateur (par exemple pour prouver que `L` est vrai dans notre problème jouet). On veut donc lui poser des questions afin de conclure sur un fait terminal. Pour cela, on procède en deux temps :

- On cherche une règle « presque déclenchable », c'est-à-dire dont toutes les prémisses sont vraies sauf une portant sur un fait qui est inconnu.
- On essaie de prouver ce fait inconnu en chaînage arrière. Si le chaînage arrière aboutit à essayer de prouver un fait observable inconnu, on pose une question à l'utilisateur sur ce fait observable. L'utilisateur peut répondre « oui », « non », ou « je ne sais pas ». Dès qu'on a une réponse « oui » ou « non » à une de nos questions, on peut ajouter un fait à la base de faits et relancer le moteur. Si l'on n'arrive toujours pas à conclure sur un fait terminal, il faut réitérer le processus. Attention à ne pas poser deux fois la même question à l'utilisateur.

Question 10 : Mettre en œuvre ce chaînage mixte et les interactions nécessaires avec l'utilisateur.

Sur notre problème jouet, en rajoutant la règle `R7 : E et L → M`, son exécution donnera :

```
?- raz.
true.

?- faits([a,c,non(d)]).
true.

?- go.
r2 : non(d) a c f
r4 : non(d) non(g) a c f
r5 : non(d) non(g) a c f b
r1 : non(d) non(g) a c f b
r3 : non(d) non(g) a c f b e
j essaie de prouver h
Est-ce que h ? (o/n/i)
|: o.
h car observé
r6 : non(d) non(g) a c f b e h l
r7 : non(d) non(g) a c f b e h l m
true .
```